J Supercomput (2012) 59:486–498
DOI 10.1007/s11227-010-0449-4

Using SPEC CPU2006 to evaluate the sequential and parallel code generated by commercial and open-source compilers

Sergio Aldea · Diego R. Llanos · Arturo González-Escribano

Published online: 10 June 2010
© Springer Science+Business Media, LLC 2010

Abstract The role of the compiler is fundamental to exploit the hardware capabilities of a system running a particular application, minimizing the sequential execution time and, in some cases, offering the possibility of parallelizing part of the code automatically. This paper relies on the SPEC CPU2006 v1.1 benchmark suite to evaluate the performance of the code generated by three widely-used compilers (Intel C++/Fortran Compiler 11.0, Sun Studio 12 and GCC 4.3.2). Performance is measure in terms of *base speed* for reference problem sizes. Both sequential and automatic parallel performance obtained is analyzed, using different hardware architectures and configurations. The study includes a detailed description of the different problems that arise while compiling SPEC CPU2006 benchmarks with these tools, an information difficult to obtain elsewhere.

Having in mind that performance is a moving target in the field of compilers, our evaluation shows that the sequential code generated by both Sun and Intel compilers for the SPEC CPU2006 integer benchmarks present a similar performance, while the floating-point code generated by Intel compiler is faster than its competitors. With respect to the auto-parallelization options offered by Intel and Sun compilers, our study shows that their benefits only apply to some floating-point benchmarks, with an average speedup of 1.2× with four processors. Meanwhile, the GCC suite evaluated is not capable of compiling the SPEC CPU2006 benchmark with auto-parallelization options enabled.

Keywords Compiler performance · Automatic parallelization · Benchmarking

S. Aldea · D.R. Llanos (✉) · A. González-Escribano
Dpto. Informática, Univ. Valladolid, Valladolid, Spain
e-mail: diego@infor.uva.es

S. Aldea
e-mail: saldeal@gmail.com

A. González-Escribano
e-mail: arturo@infor.uva.es

1 Introduction

Compilers are a critical part of any computing environment, allowing the programmers to better exploit the hardware capabilities of their systems. This work aims to help users to better understand the wide spectrum of compiler technology, evaluating three of the most popular compilers for Intel-based architectures: Intel C++/Fortran Optimizing Compiler version 11.0.074 [4, 15], Sun Studio 12 [7, 8], and GCC 4.3.2 [9]. The evaluation has been made compiling and running the SPEC CPU2006 v1.1 suite on both 32-bits and 64-bits systems, comparing the performance of these compilers on different benchmarks and hardware configurations. As long as all three compilers offer the possibility of parallelizing source code automatically, both sequential and parallel performance were evaluated. The goal of this paper is twofold: To provide results that lead to a better understanding of compiler technology and use, and to give an insight into SPEC CPU2006 benchmark suite, describing the main problems encountered while using different compiler suites.

This paper is structured as follows. Section 2 introduces the SPEC CPU2006 suite, with a brief description of its evolution and utility. Section 3 defines the execution environment, with a detailed description of the SUTs (Systems-Under-Test) used to evaluate the performance of the generated code. Section 4 provides some useful details on compiling and running SPEC 2006 in different environments and with different compiler suites. Sections 5 and 6 discuss in detail the performance of the sequential code generated by the compiler suites using the two benchmark sets that are part of the SPEC CPU2006 suite: CINT2006 and CFP2006. Section 7 describes the effect in terms of performance of the auto-parallelization flags available in Sun and Intel compilers. Section 8 describes different studies related with the goal of this work. Finally, Sect. 9 summarizes the results, providing overall ratings, and describing the main conclusions of the study.

2 SPEC CPU2006 description

SPEC (Standard Performance Evaluation Corporation) is a non-profit corporation established to maintain and endorse a standardized set of relevant benchmarks that can be applied to the newest generation of high-performance computers. SPEC was founded in 1988 by a small number of workstation vendors in search of a performance standard test that would offer some realistic and comparable results. Over time, SPEC has grown to become one of the more successful performance standardization bodies with more than 60 member companies.

To achieve these objectives, SPEC takes real-life applications programs from various fields of science and engineering, using them as benchmarks with different workloads in order to obtain performance evaluations. The first version was released in 1989 [14], and it was only composed by ten benchmarks. SPEC updates periodically the suite due to the evolution of computers and the growth and increasing complexity of application programs.

The last version, SPEC CPU2006, was released on August 24, 2006 [5], including 29 benchmarks that are encompassed in two groups: CINT2006, the integer-type applications, and CFP2006, the floating-point applications. SPEC supplies the benchmarks in the form of source code, which testers are not allowed to modify except

Table 1 Characteristics of the reference system of SPEC CPU2006. According to the specifications, one of the disks is dedicated to the operating system and the other at the code and data set of the benchmark suite

CPU model	UltraSPARC II processor
CPU characteristics	296 MHz, 2 cores, 1 chip
L1 cache size	16 KB I + 16 KB D
L2 cache size	2 MB I + D
RAM memory	2 GB
Disk subsystem	2 × 36 GB 10000 RPM SCSI
Operating system	Solaris 10 3/05
Kernel version	2.6.19
File system type	ufs
Compiler	Sun Studio 11

under very restricted circumstances. SPEC have established strict rules to run the benchmark and to report the results, to ensure that the observed level of performance can be obtained by other researchers. The benchmark also includes a tool to run and score benchmarks automatically [16].

For CPU2006, SPEC defines two type of metrics. *Base* metrics are obtained compiling and running the entire benchmark using the same compiler flags in the same order for a given language. These metrics are required for all reported results, and provide a consistent baseline for comparing performance. The *peak* metrics are optional and have less strict requirements, allowing to obtain better results using a tailored set of flags for each benchmark, while the *base* metrics have stricter guidelines for compilation. In our study, we used the *base speed* metric, since our aim is to be as fair as possible in the performance comparison of the compiler suites considered.

SPEC uses a reference machine to normalize the performance metrics used in the CPU2006 suites. Each benchmark was run and evaluated on this system to establish a reference time for that benchmark. These times are then used in the SPEC calculations. SPEC uses a Sun system, the "Ultra Enterprise 2" introduced in 1997, as the reference machine. This system has a 296 MHz UltraSPARC II processor with two cores and two GB RAM. Table 1 summarizes its characteristics.

At present, the use of the SPEC suites is fairly widespread, and many computers and processors manufacturers post SPEC results in their websites, such as Intel [2], IBM [1], Sun Microsystems [6, 17], or Fujitsu–Siemens [3].

3 Execution environments

For this work, we have used two different hardware configurations, and for one of them we use 32- and 64-bits versions of the same operating system. This leads to three different Systems-Under-Test (SUT). The first System-Under-Test (SUT1) is based on an Intel® Core™ 2 CPU E6300 processor, running a 32-bits version of the GNU-Linux operating system. SUT2 is based on the same hardware as SUT1, but it runs 64-bits version of GNU-Linux. Table 2 summarizes their characteristics. Finally, SUT3 is based on a Dual Core AMD Opteron® Processor 270 (see Table 3).

The systems described were evaluated with the entire SPEC CPU2006 v1.1 benchmark suite. The performance metric chosen was the *base speed* provided by the

Table 2 Characteristics of the Systems-Under-Test SUT1 and SUT2

CPU model	Intel® Core™ 2 CPU E6300
CPU characteristics	1.86 GHz, 1066 MHz bus. 2 cores, 1 chip
L1 cache size	32 KB I + 32 KB D on chip per core
L2 cache size	2 MB I + D on chip per core
RAM memory	3 GB (2×512 MB + 2×1 GB DDR2 667 MHz)
Operating system	Mandriva Linux Release 2007.1, 32 bits (`SUT1`)
	Mandriva Linux Release 2007.1, 64 bits (`SUT2`)
Kernel version	2.6.17-13
File system type	ext3.
Compilers	GCC 4.3.2
	Intel C++/Fortran 11.0.074 Professional Edition
	Sun Studio 12 (Sun C/C++ 5.9,
	Sun Fortran 95 8.3)

Table 3 Characteristics of the System-Under-Test SUT3

CPU model	Dual Core AMD Opteron® Processor 270
CPU characteristics	1.93 GHz, 1066 MHz bus. 4 cores, 2 chips
L1 cache size	64 KB I + 64 KB D on chip per core
L2 cache size	1 MB I + D on chip per core
RAM memory	4 GB
Operating system	Gentoo Base System release 1.12.9
Kernel version	2.6.19
File system type	ext3
Compilers	GCC 4.3.2
	Intel C++/Fortran 11.0.074 Professional Edition
	Sun Studio 12 (Sun C/C++ 5.9,
	Sun Fortran 95 8.3)

benchmark. We have chosen the use of a base metric instead of a peak metric since the former has more strict guidelines for compilation and forces to use the same compiler flags for all benchmarks, avoiding the use of tailored optimizations [5].

As its predecessors, SPEC CPU2006 provides three workload sets for each benchmark: *test*, *train,* and *reference*. The sets are ranked in order of increasing workload. The first two are workloads intended to check the correctness of the compilation and execution process, while the third one is used to evaluate performance. After running the benchmark, the SPEC CPU2006 suite generates a report with the relative performance of the SUT compared with the reference machine. To consider a report valid, it should be generated executing each benchmark three times with the test and train workloads and then three times with the reference workload. The execution times of the latter provides the final results. After the entire benchmark suite is run on the SUT, a ratio for each benchmark is calculated using the wall-clock time spent on the SUT and the time spent by the reference system, as provided by the suite in [13]. From these ratios, the suite calculates the geometric mean of 12 normalized ratios,

Table 4 Compiler and linker flags used to obtain the SPEC CPU2006 *base speed* metric used in this study. We generated 64-bit binaries for 64-bit SUTs

GCC	-O3 -funroll-loops -fno-inline-functions ftree-vectorize
INTEL	**sequential 32-bit flags:** -O3 -ipo -xT -axT -no-prec-div -funroll-all-loops -no-for-main (C and Fortran at once)
	sequential 64-bit flags: -O3 -ipo -xW -axW -no-prec-div -funroll-all-loops -no-for-main (C and Fortran at once)
	parallel flags added: -parallel
SUN	**sequential 32-bit flags:** -fast -xarch=sse3 -library=stlport4 (C++ Benchmarks except 453.povray)
	sequential 64-bit flags: -fast -xarch=sse3 -m64 -library=stlport4 (C++ Benchmarks except 453.povray)
	parallel flags added: -xautopar -xreduction

one for each integer benchmark, and the geometric mean of 17 normalized ratios, one for each floating-point benchmark.

4 Compiling and running issues

In order to compile and run the benchmark suite, each compiler needs their particular flags. Unfortunately, SPEC CPU2006 does not suggest a minimum set of flags. This makes the search for adequate flags a trial-and-error process. Moreover, as Chan et al. pointed out in 1994 [10], it is difficult to ensure that none of the flags chosen has been added to the compiler just to optimize some SPEC program, a situation not allowed by any SPEC benchmark release. Besides this, the *base speed* metric forces to use the same flags to compile the entire benchmark. After an extensive experimentation, we arrived to the flags shown in Table 4.

We found that many optimizations available benefit some benchmarks but hinder others, leading to poorer *base speed* results. Other optimizations make some compilers fail while compiling some benchmarks. A detailed description of the problems encountered follows. We believe this information is extremely useful for anyone interested in running the benchmark.

GCC compiler, 410.bwaves, 32-bits versions (SUT1) In this system, when executing 410.bwaves compiled with the `-O3` flag, the "train" input set leads to incorrect results. `-O2` flag should be used instead.

GCC compiler, 64-bits version (SUT2 and SUT3) The `-O3` flag includes by default `-finline-functions`, which expands functions during compilation. This flag makes some benchmarks fail in 64-bits systems. Therefore, we used the flag `-fno-inline-functions` to cancel the optimization.

GCC compiler, 400.perlbench The `-fno-string-aliasing` flag is needed to run this benchmark in all SUTs, so it should be included among the flags needed to obtain the *base speed* metric.

GCC compiler, auto-parallelization options The GCC compiler is unable to compile some benchmarks when adding the `-ftree -parallelizing-loops=n`

flag. This fact makes impossible to compare the effect of this feature with the corresponding features of Sun and Intel compilers, since in order to run the benchmark all applications must compile and run correctly.

Intel compiler Different problems were detected in the compilation of all the benchmarks when the `-fast` flags is used. This is because the `-static` flag is automatically included when using `-fast`. We have replaced the use of `-fast` with the use of all flags it includes, except `-static`.

Intel compiler, mixed Fortran-C applications Intel Fortran compiler does not compile correctly benchmarks that are written with a Fortran-C combination, such as 435.gromacs, 436.cactusADM and 454.calculix. The flag "nofor-main" for the Fortran compiler solves this problem. This option specifies that the main program is not written in Fortran, so it prevents the compiler to link `for_main.o` in the applications.

Intel compiler, 64-bits systems For these systems the `-xT` flag generates specialized code, enabling vectorization. In particular, according to Intel Reference Manual [4], `-xT` may generate SSE instructions for the Intel® Core™ 2 Duo Processor family. However, we have found that the use of `-xT` flag produces invalid executions. The flag `-xW`, that optimizes for Pentium™4 processor was used instead. This problem does not happen with option `-axT` that also generates non-processor specific code, although we change it to `-axW` to follow Intel recommendations.

Sun compiler, 32-bits versions (SUT1), 400.perlbench, and 416.gamess To compile these benchmarks, the `-xautopar` and `-xreduction` should not be used. This only happens with the v1.1 version of the SPEC CPU2006 benchmark, while the v1.0 version runs perfectly well with these flags.

Sun compiler, 64-bits versions (SUT2 and SUT3), 400.perlbench To compile this benchmarks, the `-xautopar` and `-xreduction` should not be used. As the preceding issue, his only happens with the v1.1 version of the SPEC CPU2006 benchmark, while the v1.0 version runs well with these flags.

Sun compiler, C++ benchmarks In these cases, SUT1 displays the following linker error with the 64-bits configuration:

```
/usr/lib64/libm.so: file not recognized: File format not
recognized
```

This error is due to a problem with the Sun linker, that is not able to find the `libm` library provided by default by our 64-bits operating system. We simply replace Sun's linker with GNU's, through a symbolic link.

Sun compiler, C++ benchmarks It was necessary to use the STLport implementation of C++ standard library (`-library=stlport4`), instead of using the library by default (`libCstd`). This change solves compilation errors in C++ benchmarks. However, in the particular case of 453.povray, this option had to be removed, using the library by default, because the STLport library causes the following error:

```
octree.cpp, line 755: Error: The function copysign must have
a prototype.
```

This library change does not invalidate the *base speed* metric obtained.

Sun compiler, auto-parallelization options When using these options with Sun compilers, it is necessary to set the `OMP_NUM_THREADS` environment variable to match the number of threads to use during the parallel execution, and the `PARAL-`

LEL variable, to set the number of available processors. In our case, we set these variables to four.

410.bwaves, 483.xalancbmk, 447.dealIII There is a run-time issue to be considered when running these benchmarks. Due to a problem with the system stack size, these benchmarks show the following error message:

```
410.bwaves: copy #0 non-zero return code (rc=0,
signal=11)
```

The solution is to increase the stack size before running the benchmark suite, through the command `ulimit -s unlimited`.

5 Sequential performance of integer benchmarks

To evaluate the relative performance of the code generated by each compiler suite, we ran the entire SPEC CPU2006 test with each compiler and each SUT considered. We start our study evaluating the relative performance of the SPEC CPU2006 integer applications. Figure 1 shows the performance of the code generated for each compiler suite for all three SUTs considered. Figure 1(a) shows the results for SUT1. As can be seen in the figure, Intel performance is much higher than the performance offered by GCC and SUN. The reason is, in part, the availability of the auto-vectorization flag in Intel, which is capable of vectorizing five loops in the 462.libquantum benchmark, while GCC is only able to do it in a single loop. Sun performance is clearly lower than Intel's, both for the geometric mean and for most of the benchmarks executed.

Figure 1(b) shows the results for the 64-bits configuration (SUT2). In this case, Sun achieves a better performance, although the differences among the compilers are not so high.

The numerical differences in terms of performance are lower in SUT3 (64-bits), and the overall performance of the machine is similar for all three compiler suites (Fig. 1(c)). It is interesting to note the lower performance of 462.libquantum for the Intel compiler, while for SUT1 and SUT2 the same benchmark runs much faster. We have found no explanation for this effect, since we have used the same compilation flags as in SUT2. Finally, it is worthwhile to remark the high performance obtained by the code generated with GCC in 464.h264ref benchmark, improving by 18% the results obtained using Intel compilers.

We conclude that, in the set of integer benchmarks of SPEC CPU2006, all three compilers perform equally well in the 64-bits environment with the optimization flags considered. Regarding the 32-bits environment, Intel shows an average improvement of 30.7% over the performance of the code generated by GCC compiler and 24.7% over Sun compiler. We remind that performance is a moving target in the field of compilers, and these results may experience variations with other SUT configurations, compiler versions or different optimizations.

6 Sequential performance of floating-point benchmarks

Performance differences are bigger for SPEC CPU2006 floating-point benchmarks. Figure 2 shows the relative performance of the execution of all 17 floating-point

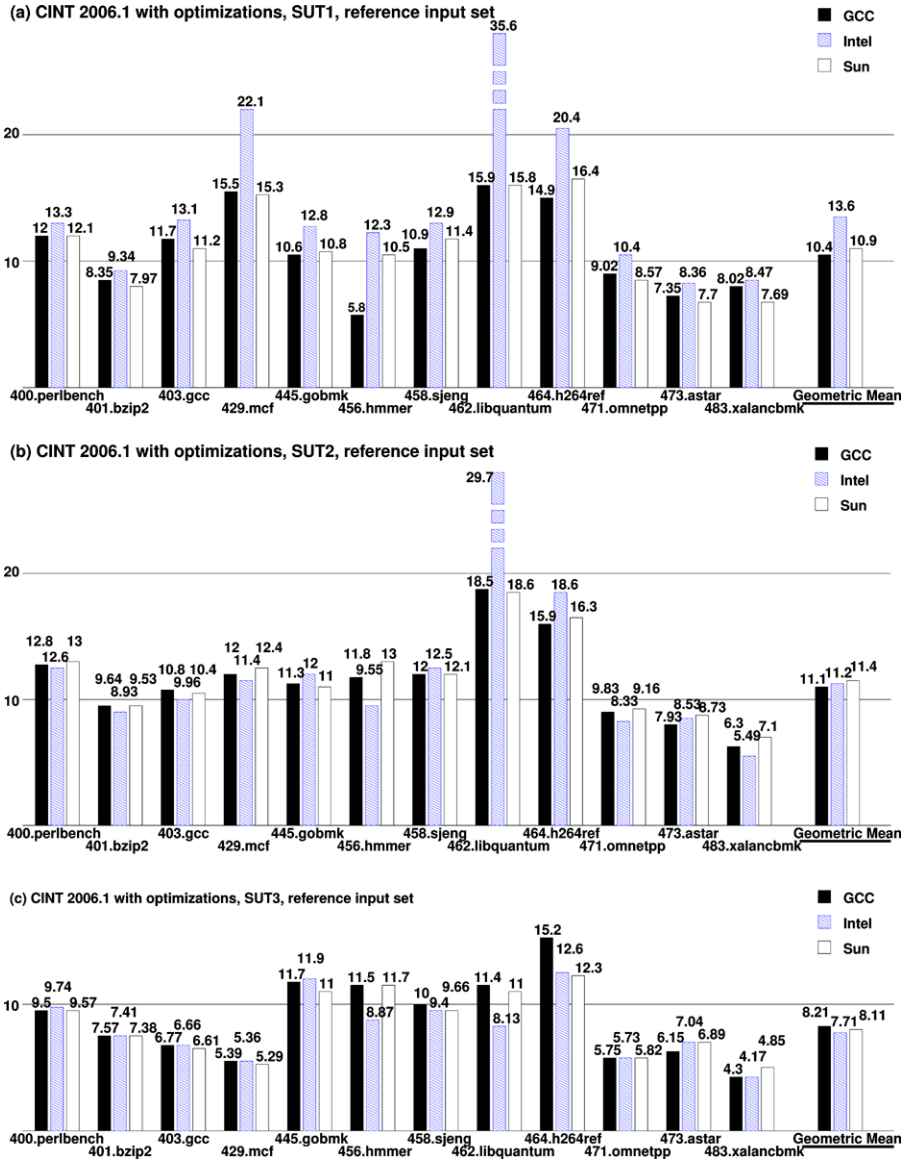

Fig. 1 Sequential performance of SPEC integer benchmarks

applications using the different compiler suites. Figure 2(a) shows the results obtained in SUT1. For this environment, Intel gets the best results in every single benchmark, surpassing the Sun compiler for more than 101% in 435.gromacs, 84% in 433.milc, or 75% in 439.cactusAMD. On the other hand, GCC obtains the worst results in 11 benchmarks, with a global performance of 7.48, 38% lower than Intel and 18% lower than Sun compilers.

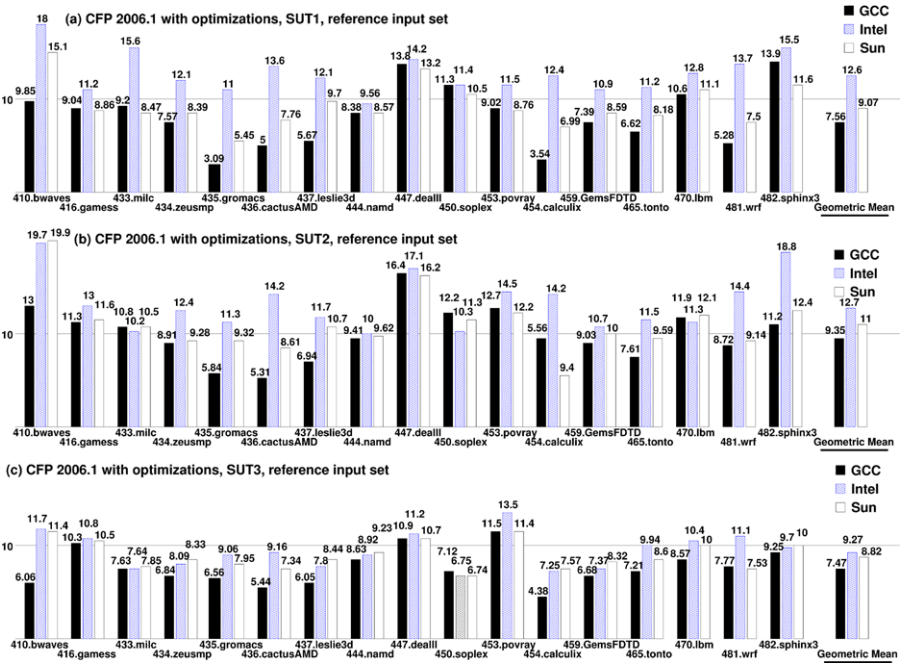

Fig. 2 Sequential performance of SPEC floating-point benchmarks

Figure 2(b) shows the results for SUT2, that is, the same hardware configuration than SUT1 but with a 64-bits operating system. Differences among compilers remain quite large. Intel obtains the best performance on 14 out of 17 benchmarks, with an average performance that is 35.8% better than the performance obtained with GCC code and 15.4% better than Sun's.

If we compare the results for SUT1 and SUT2, we observe that all three compilers improve their results when running in the 64-bits environment, while the improvement is much smaller when dealing with integer-based applications. This result is consistent with the work by Ye et al. [18] that reports a performance gain of 7% on average when running in a 64-bit address space. In the floating-point case, Intel compilers achieve an average improvement of about 7%, Sun improves by 21% and GCC improves by 25%, although the performance of the latter is well below its competitors.

Finally, Fig. 2(c) shows the relative performance of SUT3, a 64-bits system (recall Table 3). Differences are not so high in this case, although Intel performs consistently better than Sun and GCC compilers, with better results in roughly half of the benchmarks. As it happened in other cases, with rare exceptions, the use of GCC compilers lead to worse results than the use of Intel and Sun compilers, although differences are minor: 20% of slowdown with respect to the code generated by Intel and 16% with respect to Sun's.

We conclude that, in the case of the SPEC CPU2006 floating-point applications, the use of the Intel compiler suite considered leads to better performance results with

the optimization flags considered, particularly when the code runs on Intel-based architectures.

7 Parallel performance

One of the goals of this work is to evaluate the parallelization capabilities of the evaluated compilers. The following sections summarize the performance of the parallel code generated by Intel and Sun compilers. Although GCC 4.3.2 offers the possibility of exploiting loop-based parallelism through the use of the `-ftree-parallelizing-loops` flag, its use leads to compilation errors in several of the benchmarks of the SPEC CPU2006 suite. The strict rules of use of the benchmark suite prevent us to compile and evaluate separately each benchmark, so the performance of parallel GCC applications will not be considered here.

With respect to integer benchmarks, the performance results of the parallel versions generated automatically by Intel and Sun compilers are pretty much the same that for their sequential counterparts. These results that are not shown here, makes clear that the auto-parallelization capabilities offered by Intel and Sun compilers are not enough to extract any parallelism of the integer benchmarks considered. These benchmarks are in fact hardly parallelizable even by hand, and the cost of thread management is usually higher than the benefits obtained.

While differences in integer-based benchmarks are minimal, the performance gain obtained with autoparallelization mechanisms is much higher for floating-point-based applications. Figure 3 shows the results.

The relative performance of the parallel code running in SUT1 is shown in Fig. 3(a). The black lines in each bar represent the performance of the sequential execution of the same benchmark in this architecture, that is, the values already shown in Fig. 2(a). We can see that both Intel and Sun compilers present a performance gain in almost all benchmarks, with no significant slowdown in any case. The average performance obtained according to SPEC specifications shows an improvement of about 15% for Intel code and 16% for the code generated by Sun compilers. We consider these values as acceptable, taking into account that they have been obtained automatically and the parallel system is composed of just two threads. It is interesting to highlight the behavior of 436.cactusAMD, with a speedup of about 2× with two cores.

The situation changes drastically when evaluating the performance of the parallel code in SUT2, that runs a 64-bits environment. Figure 3(b) shows the results. It is surprising to discover that the Intel version of the code of several applications run *slower* in this parallel environment, while the performance of the code generated by Sun compilers is similar to SUT1. Since the purpose of this study is to evaluating the capabilities of different compilers while running SPEC benchmark code, no effort was done in migrating the applications to a 64-bits environment other than adjusting compiler flags. For this environment, Sun compilers obtains a performance gain of about 10% with respect to the sequential evaluations on SUT2.

Finally, Fig. 3(c) shows the performance results when running the benchmarks in a four-threads environment. In this case, Intel achieves a 17% improvement average

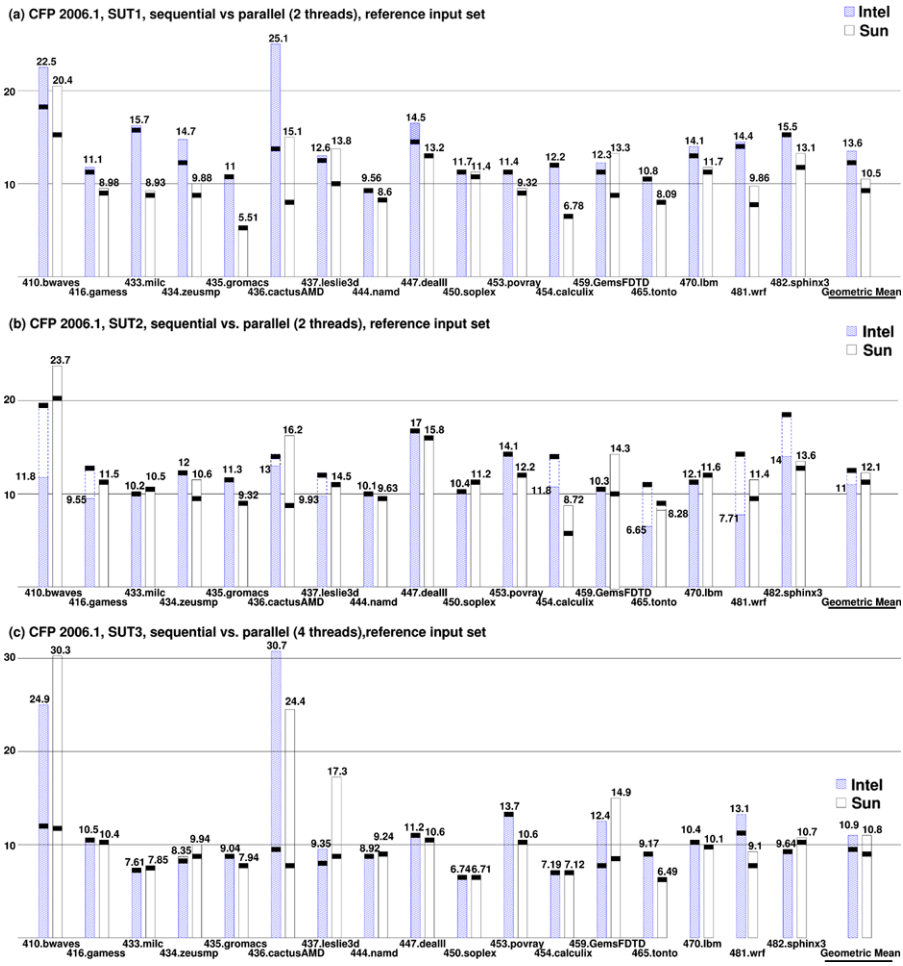

Fig. 3 Parallel performance of SPEC floating-point benchmarks. The *black box* in each bar represents the performance of the sequential version of the application in the same SUT

over the sequential code, while Sun gets about 22% improvement. Efficiency, on the other hand, drops to 29% for Intel and 30% for Sun code.

Having in mind the risks of generalizing performance results while talking about code generated by different compilers, we conclude from this results that auto-parallelization capabilities of both Intel and Sun compilers are an interesting feature that is mature enough to consistently produce a performance improvement at no cost in terms of developing effort. This fact makes these options a good starting point to better exploit the capabilities of modern multi-core systems. However, relying on these features to generate a parallel version for a many-cores architecture may not be enough to fully exploit the hardware capabilities of the system, due to the reduced performance gain obtained.

8 Related work

Surprisingly, there is a lack of comparisons about the automatic parallelization capabilities of modern compilers, using either SPEC CPU2006 or other benchmarks. However, some research has been carried out on the characterization of SPEC CPU2006 benchmark on sequential architectures. Li et al. [12] characterize the performance of SPEC CPU2006 both on Intel and AMD architectures, using GCC 4.1.2 with the `-O3` flag. Their study includes an evaluation of the level of instruction-level parallelism found. Kejariwal et al. [11] compares the behavior of both SPEC CPU2000 and SPEC CPU2006 on an Intel platform, using the Intel Optimizing Compiler, giving some hints that may guide the design of future microprocessors. Ye et al. [18] characterize the performance differences between 32- and 64-bit versions of the SPEC CINT2006 on an Intel x86-64 platform, using GCC 4.1.1. Their work shows that several X86 integer applications runs slower in 64-bit mode than in 32-bit mode, a result consistent with our study, explaining the effect based on the different instruction and data cache requested when using the different address spaces.

9 Conclusions

This work aims to help users to better understand the wide spectrum of compiler technology, evaluating three of the most popular compilers using SPEC2006 benchmarks. The work evaluates the performance of the generated code for both integer and floating-point benchmarks, not only in terms of sequential performance of the generated code, but also in terms of the availability of auto-parallelization mechanisms. All performance results obtained are available from the authors under request.

Our results show that, with respect to integer benchmarks, all three compilers perform equally well, with small differences in terms of performance that are too small to be considered significant. In the set of floating point benchmarks, differences between sequential performance of the code generated by the three compilers are more relevant. The code generated by the Intel compiler used is 39.8% faster than the code generated by GCC and 16.4% faster than the code generated by Sun compiler. Due to the constant improvements on compiler technologies, these performance differences may change in other compiler versions and/or using a different set of optimization flags.

With respect to the possibility of auto-parallelizing the code, our study concludes that this option is not useful when processing SPEC 2006 integer applications. However, in the case of the floating-point applications of SPEC 2006, the use of auto-parallelization flags allows to obtain a significant performance gain, of about 17% on average with Intel compilers and 21% with Sun compilers.

Considering not only overall results, it is worth noting that the performance gains are not uniform. For example, Intel presents a low performance in 462.libquantum benchmark in 64-bits systems, while GCC compiler generates fast code for 464.h264ref with SUT2. This prevents us to not generalize these conclusions to all applications written in a given programming language. Moreover, a tailored collection of flags for a given application may improve performance results in particular combinations of applications, compilers, and platforms. Therefore, a developer who

wants to choose between one of these compilers should evaluate their performance for his/her application and underlying architecture.

Acknowledgements This research was partly supported by the Ministerio de Educación, Spain (TIN2007-62302), Ministerio de Industria, Spain (FIT-350101-2007-27, FIT-350101-2006-46, TSI-020302-2008-89, CENIT MARTA, CENIT OASIS), Junta de Castilla y León, Spain (VA094A08), and also by the Dutch government STW/PROGRESS project DES.6397. Part of this work was carried out under the HPC-EUROPA project (RII3-CT-2003-506079), with the support of the European Community—Research Infrastructure Action under the FP6 "Structuring the European Research Area" Programme.

References

1. IBM posts SPEC CPU2006 scores for next-generation System x scalable server. ftp://ftp.software.ibm.com/eserver/benchmarks/news/newsblurb_x3850M2_speccpu_090507.pdf. IBM Corporation. Accessed 8 April 2010
2. Intel Xeon Processor performance summary. http://www.intel.com/performance/work-station/xeon/summary.htm. Intel Corporation. Accessed 8 April 2010
3. Performance and benchmarks: Performance reports for CELSIUS R640. http://www.fujitsu-siemens.com/products/deskbound/workstations/performance/celsius_r640.html. Fujitsu Siemens. Accessed 8 April 2010
4. Quick reference guide to optimization with Intel compilers version 11 Intel Software development products. http://www.intel.com/software/products/compilers/docs/qr_guide.htm. Accessed 20 January 2010
5. SPEC releases CPU2006 benchmarks. http://www.spec.org/cpu2006/press/release.html. Standard Performance Evaluation Corporation. Accessed 8 April 2010
6. Sun Fire X4450 server performance. http://www.sun.com/servers/x64/x4450/benchmarks.jsp?display=1. Sun Microsystems. Accessed 8 April 2010
7. Sun Studio 12: C user's guide. http://dlc.sun.com/pdf/819-5265/819-5265.pdf. Sun Microsystems. Accessed 8 April 2010
8. Sun Studio 12: Fortran user's guide. http://dlc.sun.com/pdf/819-5263/819-5263.pdf. Sun Microsystems. Accessed 8 April 2010
9. Using the gnu compiler collection. http://gcc.gnu.org/onlinedocs/gcc-4.3.2/gcc.pdf. Accessed 8 April 2010
10. Chan Y, Sudarsanam A, Wolfe A (1994) The effect of compiler-flag tuning on SPEC benchmark performance. SIGARCH Comput Archit News 22(4):60–70
11. Kejariwal A, Veidenbaum AV, Nicolau A, Tian X, Girkar M, Saito H, Banerjee U (2008) Comparative architectural characterization of SPEC CPU2000 and CPU2006 benchmarks on the Intel Core 2 Duo processor. In: International conference on embedded computer systems: architectures, modeling, and simulation, 2008, SAMOS 2008, pp 132–141
12. Li S, Qiao L, Tang Z, Cheng B, Gao X (2009) Performance characterization of SPEC CPU2006 benchmarks on Intel and AMD platform. In: First international workshop on education technology and computer science, 2009. ETCS '09, vol 2, pp 116–121
13. Munafo R Reference machine times for SPEC CPU2006. http://www.mrob.com/pub/comp/benchmarks/spec.html. Accessed 8 April 2010
14. Polsson K (2010) Chronology of workstation computers. http://www.islandnet.com/~kpolsson/workstat/work1987.htm. Accessed 3 June 2010
15. Schouten D, Tian X, Bik A, Girkar M (2003) Inside the Intel compiler. Linux J 2003(106):4
16. Spradling CD (2007) SPEC CPU2006 benchmark tools. SIGARCH Comput Archit News 35(1):130–134
17. Tatkar V Build high performance applications on multicore systems using sun studio compilers and tools. In: Sun TechDays India 2008, 27th–29th Feb 2008, Hyderabad, India. http://developers.sun.com/events/techdays/presentations/locations-2008/india_techdays/solaris_track/td_hyd_sun_studio_tatkar.pdf. Accessed 8 April 2010
18. Ye D, Ray J, Harle C, Kaeli D (2006) Performance characterization of SPEC CPU2006 integer benchmarks on x86-64 architecture. In: IEEE international symposium on workload characterization, 2006, pp 120–127